(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－250758
（P2000－250758A）
(43)公開日　平成12年９月14日(2000.9.14)

(51)Int.Cl.'	識別記号	ＦＩ		テーマコード'（参考）
Ｇ０６Ｆ　9/445		Ｇ０６Ｆ　9/06	４２０Ｊ	

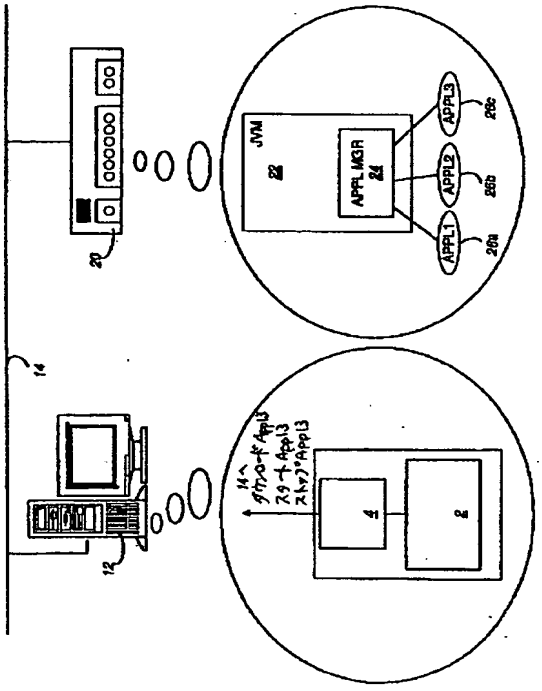
審査請求　未請求　請求項の数１　ＯＬ　（全 18 頁）

(21)出願番号　特願2000－55728(P2000－55728)
(22)出願日　平成12年３月１日(2000.3.1)
(31)優先権主張番号　２５９６１６
(32)優先日　平成11年３月１日(1999.3.1)
(33)優先権主張国　米国（ＵＳ）

(71)出願人　398038580
ヒューレット・パッカード・カンパニー
HEWLETT－PACKARD COMPANY
アメリカ合衆国カリフォルニア州パロアルト　ハノーバー・ストリート　3000
(72)発明者　フランク・ピー・ジャッジ
アメリカ合衆国コロラド州フォートコリンズ　ホルヨク・コート　525
(72)発明者　チアーチュ・ドーランド
アメリカ合衆国コロラド州フォートコリンズ　レパブリック・ドライブ　618
(74)代理人　100078053
弁理士　上野　英夫

(54)【発明の名称】　組込み機器用Ｊａｖａアプリケーションマネージャ

(57)【要約】
【課題】　組込み機器において、Ｊａｖａ仮想マシン等のハードウェア非依存型プロセッサが機器にインストールされることにより、アプリケーションが機器にダウンロードされその機器上で実行されることが可能になる。しかしながら、このＪａｖａ言語は、プログラムローダの実行もロードされたプログラムを管理する機能も提供していない。また、メモリの制約された組込み機器内でアプリケーションをダウンロードおよび管理する手段も提供していない。
【解決手段】　本願発明のＪａｖａベースのアプリケーションマネージャは、組込み機器にインストールされたＪａｖａ仮想マシン内で実行されるＪａｖａアプリケーションのダウンロード、実行およびキャッシングを制御する。さらにアプリケーションの実行の開始、初期化および停止、およびメモリ管理のための機能を提供する。



【特許請求の範囲】
【請求項１】　組込み機器において１つ以上のアプリケーションを管理するアプリケーションマネージャであって、前記組込み機器が、クラスオブジェクトを格納するアプリケーションキャッシュと、ネットワークを介してクライアントと通信するネットワークインタフェースと、該組込み機器にインストールされ該組込み機器上で実行されるＪａｖａ仮想マシンと、を備えるアプリケーションマネージャにおいて、（ａ）エントリを有し、前記エントリの各々が、前記組込み機器に現在ロードされているクラスオブジェクトを識別するクラスオブジェクトリストと、（ｂ）前記ネットワークインタフェースを介して前記クライアントから受信するアプリケーションクラスをロードし、前記アプリケーションクラスに対する新たなクラスオブジェクトを生成し、前記アプリケーションキャッシュに前記新たなクラスオブジェクトを格納し、前記新たなクラスオブジェクトを前記アプリケーションキャッシュにロードされているものとして識別するために、前記クラスオブジェクトリストに新たなエントリを付加するクラスローダメソッドと、を具備するアプリケーションマネージャ。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、組込み機器においてアプリケーションを管理するアプリケーションマネージャに関する。
【０００２】
【従来の技術】組込み機器環境では、Ｊａｖａ仮想マシン等のハードウェア非依存型プロセッサがしばしば機器にインストールされており、それによって、プログラムが機器にダウンロードされその機器上で実行されることが可能になっている。このようなシステムにより、Sun MicroSystemによるＪａｖａ（登録商標）等のハードウェア非依存型言語によって書かれたプログラムを、Ｊａｖａ（登録商標）環境をサポートするあらゆるハードウェアにダウンロードし、特定の用途のためにカスタマイズすることができる。このカスタマイズは、しばしば機器の「パーソナリティ」と呼ばれる。機器のパーソナリティを定義するために、組込み機器内で複数のアプリケーションが協調して実行されている。このように、機器を、それぞれ一意に機能するよう動的に構成することができる。例えば、冷蔵庫等の組込み機器を、その中身を自動的に追跡するようカスタマイズすることができる。食品の種類および同種の食品の命名規則が文化によって異なるように、食品の種類および命名規則を記述する新たなパーソナリティをＪａｖａ動作可能の組込み機器である冷蔵庫にダウンロードすることにより、その組込み機器の冷蔵庫を、ユーザの特定の文化に対してカスタマイズすることができる。
【０００３】
【発明が解決しようとする課題】組込み機器ドメインでは、Ｊａｖａ（登録商標）仮想マシン等のハードウェア非依存型プロセッサが機器にインストールされることにより、アプリケーションが機器にダウンロードされその機器上で実行されることが可能になる。Ｊａｖａ（登録商標）言語は、新たなプログラムを動的にロードするようプログラムを構築する基本的な機能を提供する。しかしながら、このＪａｖａ（登録商標）言語は、プログラムローダの実行もロードされたプログラムを管理する機能も提供していない。また、Ｊａｖａ（登録商標）言語は、メモリの制約された組込み機器内でアプリケーションをダウンロードおよび管理する手段も提供していない。
【０００４】いくつかの現存するＪａｖａテクノロジは、プログラムローダ・タイプの機能をサポートしている。Sun Microsystemのサーブレット（Servlet）ＡＰＩ（Application Program Interface）は、Ｊａｖａ（登録商標）動作可能ウェブ（Ｗｅｂ）サーバがサーバの機能を動的に拡張することを可能にする。サーブレットＡＰＩは、主に、ウェブサーバにおいて一般的なＣＧＩ機能に取って代るものとして生成された。サーブレットのＣＧＩタイプのプログラムに対して優れている点として、プラットフォーム非依存性、再利用可能性（オブジェクト指向技術によりＪａｖａクラスを再利用する機能）、性能効率（同じサーブレット・インスタンスが多くの要求を処理することができるサーブレットの構成可能な起動モード（ＣＧＩスクリプトに対する呼出毎に新たなプロセスを生成する必要があることとは対照的である））、および管理効率（SunのＪａｖａベースのウェブサーバはⅠ新たなサーブレット・クラスの追加、およびサーブレットの開始並びに停止を容易に管理するＪａｖａ adminアプレットを提供する）がある。
【０００５】サーブレットＡＰＩは、すべてのサーブレットが３つのメソッド、すなわちinit()（初期化）、service()（サービス）およびdestroy()（破壊）を実行するよう指令する（mandate）ことにより、サーブレットのライフサイクルを定義する。init()メソッドは、サーブレットが最初に起動される時に呼出され、service()メソッドは、クライアントからの各要求を処理するために呼出され、destroy()メソッドは、サーブレットが停止している時（すなわち、ウェブサーバがシャットダウンしている時）に呼出される。サーブレットＡＰＩは、メモリ管理を直接に管理しておらず、サーブレットを管理するための公式パッケージも提供していない。
【０００６】Hewlett Packardの組込みＪａｖａ Lab SmallWebは、ＪＶＭで実行されるＪａｖａベースのオブジェクトに対しウェブベースのインタフェースを提供する。SmallWebは、必要に応じてオブジェクトをロードすることができ、オブジェクトがその機能を（メソッド呼出により）ウェブブラウザにエクスポートする手段を提

供する。SmallWebは、明示的には、アプリケーションオブジェクトの停止またはメモリ管理機能を提供していない。更に、SmallWebには、通常ファイルシステムが必要であり、組込み環境によっては、SmallWebのオーバヘッド要求が大きすぎる場合がある。

【０００７】多くの組込みドメインにおいてＪａｖａ（登録商標）を使用する問題の１つは、Ｊａｖａ（登録商標）メモリ・サブシステムの非決定性の面である。Ｊａｖａ（登録商標）言語の非決定性メモリ管理方式により、ガーベッジ・コレクタ・メソッドgc()による参照されないオブジェクトの回収（reclamation）が可能となるが、ガーベッジ・コレクタは、これらのオブジェクトがどのようにまたはいつ回収される（reclaimed）かを指定しない。ネイティブアプリケーションは、一般に、Ｊａｖａ（商標登録）Runtime class gc()メソッドを呼出すことによりガーベッジ・コレクションを発生させる。ＪＶＭがメモリを使い尽くした場合、OutOfMemoryError（アウトオブメモリ・エラー）例外が発生する。メモリを管理する必要がないことがＪａｖａ言語の利点ではあるが、大抵の組込みアプリケーションでは、メモリがどのように管理されるかに対し、Ｊａｖａ（登録商標）言語によって現在提供されている以上に厳しい制御が必要である。そのようにより厳しい制御が必要であるのは、ローメモリ（low memory）状態で実行を続けなければならない組込みアプリケーションがあるためである。一般に、これは、継続して実行するために十分なメモリが空き状態であることを保証するよう、ネイティブの非Ｊａｖａ組込みアプリケーションにメモリマネージャを実行することによって処理される。

【０００８】従って、Ｊａｖａ動作可能な組込み機器にダウンロードを行い、そのような組込み機器におけるアプリケーションのライフタイムを制御する汎用的な方法が必要とされている。また、機器上でのアプリケーションの実行中に検出されるローメモリまたはノーメモリ（no-memory）状態を処理すると共に、プライオリティベースのアルゴリズムに従ってメモリを解放するメモリ管理ハンドラも必要とされている。

【０００９】

【課題を解決するための手段】本発明は、メモリの制約された組込み機器環境においてアプリケーションのダウンロードおよびライフサイクルを管理する新規なシステムおよび方法である。Ｊａｖａベースのアプリケーションマネージャは、組込み機器にインストールされた単一のＪａｖａ仮想マシン（ＪＶＭ）内で実行されるＪａｖａアプリケーションのダウンロード、実行およびキャッシングを制御する。Ｊａｖａ（登録商標）動作可能の組込み機器におけるクラスファイルのロード、アプリケーションの実行の開始、初期化および停止、およびメモリ管理のための機能を提供するネットワーク対応のアプリケーション・プログラム・インタフェース（ApplicationProgram Interface（ＡＰＩ））が明確化されている。本発明により、組込み機器を、使用するハードウェアのタイプに関係なく容易に再プログラムおよび管理することができる。

【００１０】本発明は、図面に関連して行われる以下の詳細な説明によってより深く理解されよう。なお、図面において、同じ要素には同じ参照番号を用いて示している。

【００１１】

【発明の実施の形態】組込み機器用の新規なアプリケーションマネージャについて、以下詳細に説明する。本発明は、Ｊａｖａ（登録商標）環境のコンテキストで説明されているが、当業者には、本発明の原理が、ハードウェア非依存型言語で書かれたコードを処理するハードウェア非依存型プロセッサを有するすべてのシステムに拡張できるということが理解されよう。

【００１２】ここで図１を参照すると、ネットワーク１４を介して組込み機器２０と通信するコンピュータシステム１２を備えたネットワークシステム１０が示されている。ネットワーク１４は、ＬＡＮまたはＷＡＮ等、従来のランドリンクされた（land-linked）有線ネットワークであっても、無線ネットワーク、またはそれらの組合せであってもよい。

【００１３】本システムにおけるコンピュータシステム１２は、クライアントとして動作し、クライアントアプリケーション２を実行する。このクライアントアプリケーション２は、クライアントインタフェース４を介して、ネットワーク１４を通じてサーバ機器（例えば、組込み機器２０）とインタフェースする。この例では、クライアントアプリケーション２は、組込み機器２０に対して要求（例えば、ダウンロード・アプリケーション３（download Appl3）、スタート・アプリケーション３（start Appl3）、ストップ・アプリケーション２（stop Appl2））を送信する。組込み機器２０は、その要求をサービスするサーバとして動作する。クライアントアプリケーション２は、コンピュータシステム１２における独自の環境で実行してもよく、あるいは、それ自身のＪａｖａ仮想マシン（ＪＶＭ）（図示せず）を含むＪａｖａ動作可能なウェブ（Ｗｅｂ）ブラウザ（図示せず）内で実行してもよい。ウェブブラウザは、主なクライアントアプリケーション・アプレットのクラスファイルの場所を指定する組込みＪａｖａアプレットを用いて、ウェブ（Ｗｅｂ）ドキュメントを解釈する。ウェブブラウザは、ＪＶＭを起動し、クライアントアプリケーション・アプレットのクラスファイルの場所を自身のクラスローダに渡す。各クラスファイルは、必要とする追加のクラスファイルの名前を知っている。これら追加のクラスファイルは、ネットワーク１４（すなわち、ネットワーク１４に接続された他のマシン）から入手されるようにしてもよく、あるいは、クライアントであるコンピュー

タシステム１２から入手されるようにしてもよい。

【００１４】組込み機器２０は、Ｊａｖａ仮想マシン（ＪＶＭ）２２がインストールされたＪａｖａ動作可能機器である。図2は、組込み機器２０をより詳細に示すブロック図である。組込み機器２０は、ＪＶＭ２２、アプリケーションキャッシュ５２およびデータキャッシュ５４を有するメモリ５０、およびネットワークインタフェース２５を備えている。ＪＶＭ２２は、クラスローダ４２および実行ユニット４６を備えている。好ましくは、ＪＶＭ２２は、バイトコードベリファイヤ４４も備えているが、メモリの制約された機器によってはこれが実行されない場合もある。本発明によれば、クラスローダ４２は、本発明のアプリケーションマネージャ２４を使用して実行される。アプリケーションマネージャ２４は、他のＪａｖａベースのプログラムのダウンロード、実行およびキャッシングを行うＪａｖａプログラムである。ＪＶＭ２２は、組込み機器２０上で実行を開始する時、アプリケーションマネージャ２４の実行を開始する。

【００１５】アプリケーションマネージャ２４は、ネットワークベースのクラスローダとして動作するサーバ機器であり、ネットワークプロトコル４８によりネットワーク１４を介してアプリケーションクラスファイル４０（以下、クラスファイル４０とする）を受取り、そのクラスファイル４０中に含まれるアプリケーションクラスを活性化することができる。当業者には周知であるように、クラスファイル４０は、Ｊａｖａ（登録商標）コンパイラによって生成されるハードウェアアーキテクチャ非依存型バイトコードを有するファイルである。このバイトコードは、ＪＶＭ２２等のＪＶＭによってのみ実行が可能である。アプリケーションマネージャ２４は、非一般的なネットワークベースのクラスローダ（ClassLoader）であり、要求毎にネットワーク１４からクラスファイル４０をロードする。一般的なクラスローダによって通常行われるようにＪａｖａ（登録商標）loadClass()（ロードクラス）メソッドによってクラスファイル４０をロードする代りに、アプリケーションマネージャ２４は、サーバソケット等のネットワークプロトコル４８で、クラスファイル４０をダウンロードするという要求を待つ。ダウンロード要求が受信されると、アプリケーションマネージャ２４はネットワーク１４からクラスファイル４０を受信し、それを解析してクラス２８ａ、２８ｂにし、Ｊａｖａ（登録商標）ClassLoader.defineClass()メソッドを呼出す。クラス２８ａ、２８ｂは、Ｊａｖａ（登録商標）ClassLoader.resolveClass()メソッドを呼出すことによって解析され、オブジェクトのインスタンス化の用意ができていることを確実にする必要がある。

【００１６】図3は、アプリケーション２６ａ、２６ｂ、２６ｃをどのようにして実行させるかを示している。図示しているように、これは、次のようにして実行される。すなわち、アプリケーションのＪａｖａクラスであるクラスオブジェクト２８ａ、２８ｂをインスタンス化し、クラスオブジェクト２８ａ、２８ｂのインスタンスであるアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃを生成し、その後アプリケーション２６ａ、２６ｂ、２６ｃを実行するためにアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃのメイン（main）メソッドを呼出す。同じＪａｖａ（登録商標）クラスの複数のインスタンスを生成することにより、同じアプリケーションの複数のインスタンスを同時に実行することができる。例えば、図3に示すように、アプリケーションオブジェクト３０ａ、３０ｂは、同じクラスオブジェクト２８ａのインスタンスであるが、アプリケーションオブジェクト３０ｃは、異なるクラスオブジェクト２８ｂのインスタンスである。従って、アプリケーション２６ａ、２６ｂは、同時に実行することができる同じアプリケーションの異なるインスタンスであるが、アプリケーション２６ｃは、完全に異なるアプリケーションである。

【００１７】アプリケーションマネージャ２４は、ダウンロード、開始、停止、問合せおよびメモリ管理機能を提供する。これらの機能をネットワーク１４を介して提供するために、組込み機器２０は、ネットワークインタフェース２５を有している。このネットワークインタフェース２５は、クライアントであるコンピュータシステム１２が組込み機器２０のアプリケーションマネージャ２４に要求を送信することができるようにするネットワークプロトコル４８を実行している。好ましい実施の形態において、ネットワークインタフェース２５は、アプリケーションマネージャ２４によって指定される汎用Ｊａｖａ言語アプリケーションプログラムインタフェース（Application Program Interface（ＡＰＩ））を用いて、アプリケーションマネージャ２４に通信する。汎用ＡＰＩにより、コンピュータシステム１２等のリモート機器に、組込み機器２０の所望の管理を指定する機能が与えられる。付録Aは、本発明の実施の形態において定義されるＡＰＩを列挙している。

【００１８】特に、アプリケーションマネージャ２４は、新たなアプリケーションがネットワーク１４を介して組込み機器２０にダウンロードされるのを可能にする。好ましい実施の形態では、これは、ＡＰＩのloadApplClass()（ロードアプリケーションクラス）メソッドによって実行される。loadAppleClass()メソッドは、Ｊａｖａのクラスファイル４０のダウンロードおよび検証、クラスファイル４０に含まれるクラスオブジェクト２８ａ、２８ｂの組込み機器２０におけるＪａｖａ（登録商標）環境へのロードを処理し、Ｊａｖａインタプリタまたはジャストインタイム（Just in Time（ＪＩＴ））コンパイラ（図示せず）を用いてアプリケーショ

ンオブジェクト３０ａ、３０ｂ、３０ｃをインスタンス化することにより、クラスオブジェクト２８ａ、２８ｂの実行を準備する。また、アプリケーションマネージャ２４は、組込み機器２０上のＪａｖａのアプリケーション２６ａ、２６ｂ、２６ｃの開始および停止を管理することにより、コンピュータシステム１２または他のネットワーク化した機器において、ユーザが、組込み機器２０にロードされたＪａｖａのアプリケーション２６ａ、２６ｂ、２６ｃの実行を制御することができるようにする。好ましい実施の形態では、これは、汎用ＡＰＩのstartAppl()（スタートアプリケーション）メソッドおよびstopAppl()（ストップアプリケーション）メソッドにより実行される。また、アプリケーションマネージャ２４は、いずれのクラスオブジェクト２８ａ、２８ｂが現在ロードされているか、いずれのクラスがアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃを有しているか、および各アプリケーションオブジェクト（インスタンス）はどのような実行状態（例えば、「初期化」、「実行中」、「終了」）にあるか等のアプリケーション情報の問合せを可能にする。これは、好ましい実施の形態では、各々applClasses()（アプリケーションクラス）メソッド、applications()（アプリケーション）メソッドおよびapplInstances()（アプリケーションインスタンス）メソッドによって実行される。ロードされたクラスオブジェクト２８ａ、２８ｂの各々の名前は、属性として、好ましくはアプリケーションマネージャ２４オブジェクト内のハッシュテーブルとして、アプリケーションリスト２３に格納される。また、アプリケーションマネージャ２４は、空きメモリの容量および総メモリの容量等、Ｊａｖａ環境からの情報の問合せを可能にする。好ましい実施の形態では、この情報の問合せは、freeMemory()（フリーメモリ）メソッドおよびtotalMemory()（トータルメモリ）メソッドを用いて行われる。また、アプリケーションマネージャ２４は、ロードされたアプリケーションがアンロードされなければならなくなる前に必要な空きメモリの容量に制限を設定する機能、およびキャッシングされたアプリケーションをアンロードすべき順序を設定する機能を含む、メモリ管理機能を提供する。これは、好ましい実施の形態では、setFreeMemoryLimit()（セットフリーメモリリミット）メソッドおよびsetFreeAppsFirst()（セットフリーアプリケーションファースト）メソッドを用いて実行される。空きメモリの制限およびアプリケーションをアンロードすべき順序については、getFreeMemoryLimit()（ゲットフリーメモリリミット）メソッドおよびgetFreeAppsFirstPolicy()（ゲットフリーアプリケーションファーストポリシー）メソッドを用いて問合わせることができる。この情報は、アプリケーションマネージャ２４オブジェクト内のハッシュテーブルに、各アプリケーションリスト２３エントリ毎に別々のフィールドとして、または別々のアンロードプライオリティリスト２１として格納される。

【００１９】ネットワークプロトコル４８を用いて、ネットワークインタフェース２５を介するアプリケーションマネージャ２４に対するネットワークアクセスが提供される。ネットワークプロトコル４８は、いくつかの他の実施の形態のうちの任意のものによって実現するようにしてもよい。

【００２０】第１の実施の形態では、ネットワークプロトコル４８は、Ｊａｖａ（登録商標）リモートメソッド呼出（Remote Method Invocation（ＲＭＩ））を用いて実現される。ＪａｖａＲＭＩにより、他のＪＶＭに常駐しているオブジェクトにおいて直接のメソッド呼出を行うことができる。一般に、ＲＭＩには、オブジェクトのシリアライズ、識別およびメソッドのディスパッチをサポートするために大容量のメモリが必要である。組込み機器が、ＲＭＩネームサーバの実行のサポートを含む実行中のＲＭＩの高メモリオーバヘッドをサポートする場合、ＲＭＩを選択して実現するということは、組込み機器２０およびアプリケーションマネージャ２４と対話（interact）するクライアントアプリケーションがＪａｖａベースである場合に適している。他の分散オブジェクトメカニズム（例えば、ＣＯＲＢＡ、ＤＣＯＭ）とＲＭＩとの間には、ブリッジが存在し、それにより、クライアントにおいて他の言語で同様に書込みことが可能となる。アプリケーションマネージャ２４は、ＲＭＩを使用してリモートクライアント１２から直接の呼出を受入れることにより、アプリケーション２６ａ、２６ｂ、２６ｃをロードし、開始し、管理することができる。ＲＭＩベースのアプリケーションマネージャインタフェースの簡単な例を、付録Ｂに示す。

【００２１】第２の他の実施の形態では、ネットワークプロトコル４８はネットワークソケット（例えば、ＴＣＰ／ＩＰソケットプロトコル）を用いて実現される。クライアント１２（コンピュータシステム１２）側で同様にＪａｖａ（登録商標）が使用される場合、ネットワークプロトコル４８がクライアント１２（コンピュータシステム１２）側とサーバ（組込み機器２０）側との両方で一貫していることを保証するために、Ｊａｖａクラス内にネットワークプロトコル４８をラップする（wrap）ことは有益である。クライアント１２（コンピュータシステム１２）またはアプリケーションマネージャ２４が停止する可能性のあるネットワークの読出のブロック化を、ネットワークプロトコル４８が実行しないことを保証するために、好ましくは、Ｊａｖａ（登録商標）Socket.setSoTimeout()メソッドにより、またはＪａｖａ（登録商標）InputStream.available()メソッドを用いてブロック化無しにどれくらいのデータが読出されるかを判断することにより、read()（読出し）動作のための待ちのタイムアウトが設定されている。

【００２２】かかるネットワークソケットプロトコル方

式の１つの実現例では、呼出されるメソッドを識別する単一バイトと、それに続く、ラッパ（wrapper）によりその特定のメソッドに対してデコードされる追加のデータとが、送信される。例えば、クラスファイル４０をダウンロードするために、クライアント１２（コンピュータシステム１２）は、その要求がダウンロードのためのものであることを指定する「Ｄ」キャラクタバイトと、それに続く送信されているデータの全体長と、それに続く組込み機器２０のネットワークアドレスとを送信する。他の実現例では、実際のデータはナル終了キャラクタのクラス名であり、そのクラスがメインメソッドを含むか否かを指定するバイトがその後に続き、更にクラスデータを含むバイトがその後に続く。プロトコルのデコードを実行するために、仮想デコードメソッドをサポートするオブジェクトモデルが生成される。また、クライアント１２（コンピュータシステム１２）にメソッド呼出の結果を返すことを、使用されている同様のプロトコルを用いてカプセル化してもよい。このように、要求から結果が非同期に返され、クライアント１２（コンピュータシステム１２）は実際に、結果に対しネットワークサーバとして動作することができる。クライアント１２（コンピュータシステム１２）が１つの組込みＪＶＭ２２に通信する（talk）のみであり、結果を待ってアイドル状態である場合、クライアント（コンピュータシステム１２）が、結果が同期して戻ってくるのを待つ（タイムアウト有りで）、より単純な方式を使用することができる。

【００２３】好ましい実施の形態では、アプリケーションマネージャ２４は、Ｊａｖａネットワークサーバ・アプリケーションとして実現され、Ｊａｖａ（登録商標）リモートメソッド呼出（ＲＭＩ）プロトコル、ウェブがホストする要求のためのハイパーテキスト転送プロトコル（ＨＴＴＰ）、またはＴＣＰ／ＩＰソケットによるアプリケーションレベルプロトコル等のネットワークプロトコル４８にコード化される要求を受入れる。上述したように、ネットワークプロトコル４８の要求の１つのタイプは、アプリケーションのＪａｖａのクラスファイル４０のダウンロードである。好ましい実施の形態では、クラスファイル４０のダウンロードは、loadApplClass()メソッドまたはloadAndInit()（ロードおよび初期化）メソッドを呼出すことによって実行される。上述したように、クラスファイル４０は、ＪＶＭ２２がＪａｖａクラスオブジェクト（アプリケーションオブジェクト３０ａ、３０ｂ、３０ｃ）を生成するために使用するバイナリのＪａｖａバイトコードストリームである。アプリケーションクラスファイル４０のメモリ５０へのダウンロードに加えて、アプリケーション２６ａ、２６ｂ、２６ｃが継承するあらゆる基底クラス、またはそのクラスファイル４０に含まれるアプリケーションクラス２８ａ、２８ｂとのインタフェースもまた、メモリ５０のアプリケーションキャッシュ５２にダウンロードされる。アプリケーションマネージャ２４がクラスファイル４０のメモリ５０へのキャッシングを可能にするため、ネットワークプロトコル４８は、startAppl()（スタートアプリケーション）メソッド呼出が後に続くinitAppl()（初期化アプリケーション）メソッドにより、メモリ５０に常駐している既にダウンロードされているクラスファイルの実行を可能にする。クラスファイル４０がダウンロードされると、Ｊａｖａ（登録商標）ClassLoader ＡＰＩを用いて、クラスファイル４０に関連するクラスオブジェクト２８ａ、２８ｂがインスタンス化される。ClassLoader.defineClass()メソッドは、クラスファイルのバイトアレイからクラスオブジェクト２８ａ、２８ｂを構成する。クラスは、定義されると、クラスのリンク化を実行するために解析されなければならず、それによってインスタンスオブジェクトが生成されメソッドが呼出される。解析プロセスは、Ｊａｖａ（登録商標）ClassLoader.resolveClass() ＡＰＩを呼出すことによって起動される。

【００２４】クラスオブジェクト２８ａ、２８ｂがロードおよび定義されると、一例としての実施の形態ではＪａｖａ（登録商標）Class.newInstance() ＡＰＩにより、クラスオブジェクト２８ａ、２８ｂのインスタンスであるアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃが生成される。アプリケーションオブジェクト３０ａ、３０ｂ、３０ｃは、アプリケーション２６ａ、２６ｂ、２６ｃを開始するために所望のメソッドを呼出す必要がある。newInstance()（ニューインスタンス）を呼出すために、クラスオブジェクト２８ａ、２８ｂは、属性をとらないデフォルトのコンストラクタを有していなければならない。

【００２５】アプリケーション２６ａ、２６ｂ、２６ｃは、予め生成されたアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃの所望のメソッドを呼出し、そのメソッドに属性を渡すことによって実行される。アプリケーションマネージャ２４は、オブジェクトの呼出すメソッド、および渡す属性（あるとすれば）が何であるかを知らなければならない。これは、いくつかの方法のうちの１つによって実行される。

【００２６】１つの実施の形態では、すべてのアプリケーションは、呼出すメソッドを指定するＪａｖａインタフェースを実行する必要がある。例としてのインタフェースを以下に示す。

```
public interface ApplBase
        {
            public void main(InetAddress 0);
        }
```

このインタフェースは、アプリケーション２６ａ、２６ｂ、２６ｃの実行時に呼出される最初のメソッドであるインスタンスメソッド「main()」を指定する。それがイ

ンスタンスメソッドであるため、クラスオブジェクト２８ａ、２８ｂのアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃは、アプリケーションマネージャ２４により、main()を実行する前に生成されなければならない。main()のみが呼出されているため、このオブジェクトはアプリケーションマネージャ２４のＪａｖａ（登録商標）AppIBaseオブジェクトとしてもよい。この例では、main()メソッドは、クラスをダウンロードしたクライアント（コンピュータシステム１２）のネットワークアドレスを含むInetAddress引数を受取る。このように、アプリケーションは、必要ならば周知のポートによりクライアント（コンピュータシステム１２）に戻る方向に通信することができる。

【００２７】別の実施の形態では、ＪＶＭ２２が例えばＪａｖａ（登録商標）ＲＭＩによりリフレクションをサポートする場合、メソッド名がアプリケーションマネージャ２４にダウンロードされ、リフレクションを用いて所望のメソッドのアプリケーションクラスが探索される。クラスオブジェクト２８ａ、２８ｂがロードされると、所望のメソッドを見つけるためにそのクラスのＪａｖａ（登録商標）getDeclaredMethod()が呼出される。getDeclaredMethod()はMethodオブジェクトを返すものであって、Methodオブジェクトは、invoke()（呼出）を呼出すことにより、そのメソッドを実行するために使用することができる。

【００２８】更に他の実施の形態では、すべてのアプリケーションが継承する元となる共通の基底クラスから拡張されるメソッドが呼出される。

【００２９】アプリケーションマネージャ２４は、実行中となると、アプリケーション２６ａ、２６ｂ、２６ｃの新たな実行状態を記録する。各クラスオブジェクト２８ａ、２８ｂは、そのクラスの存在する各インスタンスを識別する各々のインスタンスリスト２９ａ、２９ｂを有している。インスタンスリスト２９ａ、２９ｂの各エントリは、現アプリケーション２６ａ、２６ｂ、２６ｃのライフサイクルの間にアプリケーションマネージャ２４によって更新される実行状態変数を含む。

【００３０】アプリケーションマネージャ２４は、アプリケーション２６ａ、２６ｂ、２６ｃで発生するＪａｖａ（登録商標）例外またはエラーのすべてを処理する。そうしなければ、ＪＶＭ２２は終了することとなる。アプリケーションマネージャ２４は、アプリケーションが終了すると、そのアプリケーション２６ａ、２６ｂ、２６ｃがもはや実行していないということを（インスタンスリスト２９ａ、２９ｂにおいて識別されるインスタンスに対応する実行状態を更新することにより）記録し、クラスオブジェクト２８ａ、２８ｂがメモリ５０内にキャッシングされるものでない場合、そのクラスオブジェクト２８ａ、２８ｂがガーベッジ・コレクトされるようにする。

【００３１】クラスオブジェクト２８ａ、２８ｂのキャッシングは、性能をより優れたものとする本発明の独特の特徴である。アプリケーション２６ａ、２６ｂ、２６ｃが実行される毎にアプリケーションクラス（クラスオブジェクト）をダウンロードする代りに、アプリケーションマネージャ２４は、好ましくは、デフォルトでクラスオブジェクト２８ａ、２８ｂをキャッシングし、unloadAppl()（アンロードアプリケーション）メソッドを用いて明示的に要求される場合、またはメモリ管理ハンドラ２７がローメモリまたはノーメモリ状態の結果としてアンロードすべきクラスを選択する場合にのみ、クラスオブジェクト２８ａ、２８ｂをアンロードする。好ましい実施の形態では、メモリ管理ハンドラ２７によって現在ロードされているクラスがアンロードされる順序は、setFreeAppsFirst()メソッドを用いて設定される。setFreeAppsFirst()メソッドにより、クライアントは、ローメモリまたはノーメモリ状態の場合にアンロードされるアプリケーションの順序を設定または変更することができる。これは、クラスをアンロードするためのアプリケーションの順位付けを指示するアプリケーションリストにおいて、各エントリにアンロードプライオリティフィールドを付加する等、いくつかの方法のうちの任意の方法によって達成することができる。アプリケーションマネージャ２４は、クラスをキャッシングすることにより、クラスに対する参照（reference）を保持し、それによって、Ｊａｖａ仮想マシン２２に、クラスオブジェクト２８ａ、２８ｂに対するガーベッジ・コレクションを行わせない。そして、クラスオブジェクト２８ａ、２８ｂを再利用して、クラスオブジェクト２８ａ、２８ｂの新たなインスタンス３０ａ、３０ｂ、３０ｃを生成し、アプリケーション機能を再実行することができる。

【００３２】また、アプリケーションマネージャ２４は、好ましくは、アプリケーションが終了および／またはアンロードされた後であっても、メモリ５０内のデータキャッシュ５４へのアプリケーションデータのキャッシングを可能にする。例として、電子テストドメインにおけるＪａｖａアプリケーションがある。アプリケーションが最初に実行されるとセットアップ情報を保存することができ、それによって、そのアプリケーションを将来実行する場合により高速に実行することができる。この例では、アプリケーションマネージャ２４は、テストシステム較正（calibration）情報をキャッシングすることができる。更に、テストシステム較正情報をデータキャッシュ５４にキャッシングすることにより、アプリケーションマネージャ５４はデータへの参照を保持し、それによってＪａｖａ仮想マシンにデータをガーベッジ・コレクトさせないようにする。従って、この方法でのデータキャッシングにより、同じかまたは異なるアプリケーションの後続する実行のために情報を保存することができる。

【００３３】アプリケーションマネージャ２４は、アプリケーション２６ａ、２６ｂ、２６ｃをローメモリまたはノーメモリ状態で実行を続けるよう試みる。ＪＶＭ２２が、アプリケーション２６ａ、２６ｂ、２６ｃの実行中にメモリを使い尽くした場合、OutOfMemoryErrorエラーが生成される。アプリケーションマネージャ２４は、ローメモリまたはノーメモリ状態を処理するメモリ管理ハンドラ２７を有している。好ましい実施の形態では、メモリ管理ハンドラ２７は、ＪＶＭ２２によって生成されるOutOfMemoryErrorの発生により実行するようトリガされる。１つの実施の形態では、メモリ管理ハンドラ２７は、そのアプリケーションキャッシュからクラスオブジェクト２８ａ、２８ｂおよびアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃを適切にダンプすることにより作用する。キャッシュされたクラスオブジェクト２８ａ、２８ｂおよびアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃをアンロードする順序は、setFreeAppsFirst()メソッドを用いて設定される。あるいは、その順序は、メモリ管理ハンドラ２７自身においてコード化された予め決められたアルゴリズムに従って決定される。アプリケーションマネージャ２４は、Ｊａｖａ（登録商標）Runtime.gc()メソッドを呼出すことにより、アンロードされたクラスオブジェクト２８ａ、２８ｂおよびアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃが回収（reclaim）可能であることをＪＶＭ２２に通知する。アプリケーションキャッシュ５２からクラスオブジェクト２８ａ、２８ｂをアンロードすることにより、そのクラスオブジェクト２８ａ、２８ｂに関連するアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃが同様に終了およびアンロードされる。従って、アプリケーション実行の起動速度が重要である場合、クラスオブジェクト２８ａ、２８ｂをアンロードする前にアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃをアンロードすることが好ましい。一方で、特定のアプリケーションを無関係に実行させておくことが重要である場合、その特定のアプリケーションを実行させておくために十分なメモリを解放するべく、それらオブジェクトに関連するクラスと共に別のクラスのすべてのアプリケーションオブジェクトをアンロードすることが好ましい。アプリケーションを終了させることについての解決法は、ローメモリ状態に応答して別々にメモリレベルを監視するか、またはアプリケーションの結果をチェックポイントすることによりアプリケーションがチェックポイントの後に継続を再開することができるようにすることである。これらの方法により、実行中のアプリケーションは割込みされずに、または少なくとも可能な限り割込みされずに継続することができる。

【００３４】アプリケーションマネージャ２４は、好ましくは、メモリ管理機能を提供する。１つの実施の形態では、アプリケーションマネージャ２４は、アプリケーションキャッシュ５２およびデータキャッシュ５４各々におけるクラスオブジェクト２８ａ、２８ｂ、アプリケーションオブジェクト３０ａ、３０ｂ、３０ｃおよびグローバルデータを参照する。それによって、それらの関連するオブジェクトがガーベッジ・コレクトされないことが保証される。

【００３５】他の実施の形態では、メモリ管理ハンドラ２７は、連続的に空きメモリレベルを継続して監視し、ローフリー（low-free）メモリまたはノーフリー（no-free）メモリ時には、そのキャッシュ５２、５４からオブジェクトを除去し、必要に応じてガーベッジ・コレクションをトリガして更にメモリを解放する。上述したように、ガーベッジ・コレクションは、アプリケーションキャッシュ５２またはデータキャッシュ５４からアイテムを除去した後にＪａｖａ（登録商標）Runtime.gc()メソッドを呼出すことによってトリガされる。アプリケーションマネージャ２４を実行する場合、それがローメモリ状態においてメモリ５０をどのように管理するかについて、選択を行わなければならない。１つの選択は、ローフリーメモリが発生した時に、アプリケーションキャッシュ５２またはデータキャッシュ５４のいずれのキャッシュから最初にアイテムが除去すべきかということである。大抵のケースでは、アプリケーションオブジェクト３０ａ、３０ｂ、３０ｃおよび関連するクラスオブジェクト２８ａ、２８ｂを除去し、必要に応じてそれらを再ダウンロードする方が容易である。この判断は、グローバルデータを再生成するためにかかる長い時間に基づいている。

【００３６】アプリケーションおよびグローバルデータオブジェクトは、好ましくは、キャッシング方式が正確に作用するために可能な限り別々であるように設計されている。グローバルデータクラスがクラスオブジェクト２８ａ、２８ｂのメンバとして参照される場合、クラスオブジェクト２８ａ、２８ｂもまた参照されないということがない限り、データクラスが参照されないということでメモリは解放されない。グローバルデータクラスがクラスオブジェクト２８ａ、２８ｂを参照するという逆の場合によっても、同じ結果となる。クラスオブジェクト２８ａ、２８ｂは、クラスメンバではなくメソッドにおいてのみグローバルデータクラスを参照すべきである。好ましくは、グローバルデータオブジェクトは、クラスオブジェクト２８ａ、２８ｂを参照することはない。

【００３７】図４は、アプリケーションキャッシュ５２の管理の一例としての実施の形態を示す動作フローチャートである。アプリケーションマネージャ２４は、クライアントであるコンピュータシステム１２から要求を受信する（４０２）。要求の実行により空きメモリを使用すると（例えば、loadAppl()、loadAndInit()、InitApp1()）、受信した要求の実行によりローメモリまたはノ

ーメモリ状態となったか否かを判断する（４０４）。その場合、アプリケーションキャッシュ５２からアンロードするクラスオブジェクト２８ａ、２８ｂおよび／またはアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃを選択する（４０６）。そして、選択したオブジェクトをアプリケーションキャッシュ５２からアンロードする（４０８）。受信した要求がloadAppl()またはloadAndInit()要求である場合、ネットワーク１４からクラスオブジェクトをロードし定義する（４１０）。要求がloadApp()要求である場合、その要求は完了する。要求がloadAndInit()要求またはInitAppl()要求である場合、アプリケーションオブジェクト３０ａ、３０ｂ、３０ｃをインスタンス化し（４１２）、その要求は完了する。要求がstartAppl()要求である場合、概してアプリケーションオブジェクト３０ａ、３０ｂ、３０ｃのmain()メソッドを呼出すことにより、アプリケーションを開始する（４１４）。要求がstopAppl()要求である場合、アプリケーションを停止する（４１６）。要求がunload()要求である場合、クラスオブジェクト２８ａ、２８ｂをアンロードする（４１８）。要求がsetFreeMemoryLimit()要求である場合、ローメモリ状態の判断の基準となる空きメモリ閾値を設定する（４２０）。要求がsetFreeAppsFirst()要求である場合、ローメモリまたはノーメモリ状態が検出された時にアンロードのためにアプリケーションクラス２８ａ、２８ｂおよびオブジェクトクラス３０ａ、３０ｂ、３０ｃが選択される順序を設定する（４２２）。要求が問合せ（例えば、いずれのアプリケーションクラスがロードされるか（applClasses()）、いずれのアプリケーションが初期化されるか（applications()）、いずれのアプリケーションが現在実行中であるか（applInstances()）、どのくらい使用可能なメモリが存在するか（freeMemory()）、どのくらい総メモリが存在するか（totalMemory()）、現在の空きメモリ制限閾値は何であるか（getFreeMemoryLimit()）、またはクラスおよびアプリケーションオブジェクトをアンロードするための現在の順序は何か（getFreeAppsFirstPolicy()））、その問合せを実行し要求されたパラメータを返す（４２４）。

【００３８】ローメモリまたはノーメモリ状態が存在すると決定した場合（４０４）、一例としての実施の形態では、アプリケーションマネージャ２４は、アンロードするよう選択されているオブジェクトに対する参照を除去し、Ｊａｖａ（登録商標）Runtime.gc()メソッドを呼出すことにより、参照されないオブジェクトがアプリケーションキャッシュ５２から除去されるようにする。上述したように、Ｊａｖａ（登録商標）Runtime.gc()メソッドによって参照されないクラスのみをガーベッジ・コレクトすることができるため、参照されないクラス２８ａ、２８ｂのみがアプリケーションキャッシュ５２から除去される。ＪＶＭ２２の実現例によっては、そのクラスのClassLoaderオブジェクトもまたガーベッジ・コレクトされるクラスとして参照されてはならないものとしてもよい。

【００３９】図５は、本発明を採用するシステム５００の一例として実現例を示すクラス図である。図５の一例としての実現例において定義および使用されるクラスの各々の記述を、以下の表１に示す。表２には、図５の一例として実現例において使用されるクラスの各々において定義される主な属性の記述が含まれている。図５に示すように、クラスApplMgr５０２は、クラスApplication５１０の１対多のインスタンスを管理する。ApplMgr５０２は、クライアントクラスApplClient５０８とインタフェースするために、クラスClientProtocol５０６に通信するクラスServerProtocol５０４を使用する。表１に記述しているように、ApplMgr５０２は、アプリケーションマネージャ２４を実現するクラスである。Application５１０は、単一のアプリケーションをモデル化している。ServerProtocol５０４およびClientProtocol５０６は、クライアントインタフェース４を実現する。それらは共に、アプリケーションマネージャクラスApplMgr５０２とクライアントクラスApplClient５０８との間のネットワークプロトコル４８をカプセル化する。

【００４０】表１

クラス：ApplMgr
記述：アプリケーションマネージャ全体のクラス
クラス：Application
記述：単一のアプリケーションをモデル化
クラス：ServerProtocol,ClientProtocol
記述：ApplMgrによってサポートされるインタフェースを表すApplMgrクラスとApplClientクラスとの間のネットワークプロトコルをカプセル化
クラス：ApplBase
記述：ApplMgrによって使用されるためにApplicationが定義しなければならないメソッドを定義するインタフェース
クラス：ApplClient
記述：ApplMgrクラスと対話する一例としてのクライアント側アプリケーション

【００４１】表２

名前：ApplMgr.idata
カテゴリ：属性
記述：Applicationクラスのライフタイムを超えて存続する必要のあるグローバルデータ値のハッシュテーブル。getメソッドおよびsetメソッドによりアクセスされる。
名前：ApplMgr.iappls
カテゴリ：属性
記述：ApplMgrが管理するよう指示されたApplicationオブジェクトの集まり
名前：Application.iclasses

カテゴリ：属性
記述：このApplicationに関連するクラスのハッシュテーブル。クラスは名前によって一意に識別される。クラスの階層構造を用いてアプリケーションを生成することができるが、それにはハッシュテーブルの使用が必要である。
名前：Application.iappls
カテゴリ：属性
記述：現在実行中のアプリケーションのインスタンスの集まり。インスタンスは、クラス名によって識別され、インスタンスカウンタは生成される各インスタンスについてインクリメントされる。
名前：Application.imainClass
カテゴリ：属性
記述：実行されるmain()ルーチンを含むクラス。実行されるアプリケーションオブジェクトは、このクラスから生成される。
名前：ServerProtocol.isock
カテゴリ：属性
記述：ApplMgrとの通信のために使用されるServerSocket。
名前：ServerProtocol.iapplMgr.
カテゴリ：属性
記述：ServerProtocolはあらゆる要求をデコードし、その要求を処理するためにApplMgrに対し適当な呼出を行う。あるいは、ServerProtocolに代えて、ApplMgr.に対する直接のＲＭＩ呼出でもよい。
名前：ClientProtocol.isock
カテゴリ：属性
記述：ClientProtocolがServerProtocolと通信する手段であるソケット。
【００４２】図６は、新たなクラスのロードを実行するためのクラス間の通信を示すシステム５００のブロック図である。表３は、その初期の仮説、ロード動作の結果、およびロードを実行するために必要なエージェントを指定している。ここに示すように、アプリケーションマネージャクラスであるApplMgr５０２は、組込み機器２０のＪＶＭ２２において実行中であり、かつ、ユーザが、クライアントであるコンピュータシステム１２で実行中のApplClient５０８に対し新たなクラスを組込み機器２０にダウンロードするよう命令することにより、ダウンロードを要求したものと仮定する。
【００４３】表３
仮説：ApplMgrは既にＪＶＭで実行中である。
結果：アプリケーションに関連するクラスは、ＪＶＭにロードされており、「メインクラス」のインスタンスが生成され、この新たなインスタンスにおいてinit()メソッドが呼出される。
エージェント：
User＝ApplClientに対し新たなアプリケーションクラスをApplMgrを実行している組込み機器にダウンロードするよう命令
ApplClient＝アプリケーションクラスをダウンロードするホスト側プログラム
ClientProtocol＝クライアントのためのネットワーク通信を処理するホスト側クラス
ServerProtocol＝ApplMgrのためのネットワーク通信を処理するサーバ側クラス
ApplMgr＝アプリケーションマネージャクラス
Application＝ダウンロードされたアプリケーションをモデル化するアプリケーションクラス
図６に示すように、ダウンロード動作は、クライアントユーザクラスApplClientU５１６等のクライアントユーザプリケーションからクライアントアプリケーションApplClient５０８へのユーザ要求６０２によって開始する。この例では、クライアントユーザクラスであるApplClientU５１６は、「Ｄ」キャラクタと、それに続くクラスデータを含むファイルの名前と、更にそれに続くクラスデータを送信すべきネットワークノードとを送信するプロトコルに従うことにより、ダウンロード要求を生成する。この例では、クラスデータは「test」と名づけられ、組込み機器２０のネットワークノードアドレスは、「ＨＰ」と名づけられている。ユーザ要求６０２により、ApplClient５０８が実行され、クラス「test」がノード「ＨＰ」にダウンロードされる。ApplClient５０８は、「test」に含まれるクラスデータをメモリに読出し、その後ClientProtocl.loadClass()メソッドを呼出してクラスをダウンロードする（６０４）。ClientProtocol５０６は、ノード「ＨＰ」上のServerProtocolへのネットワーク１４コネクションを形成し、ネットワーク１４を介してクラスデータファイル「test」に関連するクラスデータを送信する（６０６）。ServerProtocol５０４は、要求およびクラスデータをデコードし、その後ApplMgr.loadClass()メソッドを呼出してクラスをロードする（６０８）。ApplMgr５０２は、必要ならばクラスに対してApplicationオブジェクトを生成する（６１０）（すなわち、クラスは既にアプリケーションキャッシュにキャッシングされていない）。そして、ApplMgr５０２は、Application５１０のloadClass()メソッドを呼出し（６１２）、それにクラス「test」に関連するすべてのクラスデータを渡す。新たなクラスオブジェクトが、ApplicationsのApplMgrキャッシュに付加され（６１４）、それが管理するそのアプリケーションオブジェクトへのクラス「test」の付加を反映するようにアプリケーションリストApplMgr.iapps２３が更新される（６１４）。ロードされているクラス「test」がメインクラスである場合、Application５１０はクラスを定義して解析し、クラスのインスタンスを生成し、インスタンスのinit()メソッドを呼出す（６１８）。新たなアプリケーションオブジェクトが、クラスキャッシュに付加され

（６１６）、その新たなアプリケーションオブジェクトの付加を反映するようにインスタンスリスト２９が更新される（６１６）。真の結果は、クラスのロードが成功したことを通知してApplClient５０８に戻される（６２０）。

【００４４】実現例において、必要ならば、ServerProtocol５０４は、各要求を処理するための別々のスレッドを生成し、複数の要求が同時に処理されるようにする。このようにして実現された場合、ApplMgr５０２およびApplication５１０によって実現される方法は、適切に同期しなければならない。

【００４５】Application５１０が複数のクラスから構成されている場合、各クラスに対しloadClass()要求が作成されることとなる。これには、ApplMgr５０２が現存のApplication５１０にクラスを配置および加えることができる必要がある。アプリケーションは、メインクラスの名前と同一の一意の名前によって識別される。慣習的に、基底クラスおよびインタフェースは、メインクラスをロードする前にロードされる。これにより、メインクラスのインスタンスが正確に生成されることが保証される。別の方法は、アプリケーションのインスタンスを生成するために新たなApplMgrメソッドを生成するというものである。これにより、同じクラスの複数のアプリケーションが同時に実行されることが可能になる。

【００４６】図7は、アプリケーションの開始を実行するためのクラス間の通信を示すシステム５００のブロック図である。表４は、初期の仮説、開始動作の結果、および開始を実行するために必要なエージェントを指定している。ここに示すように、アプリケーションマネージャクラスであるApplMgr５０２は、組込み機器２０のＪＶＭ２２において実行中であり、かつ、ユーザが、そのユーザのマシンで実行中のApplClient５０８に対し組込み機器２０でアプリケーションを開始するよう命令することにより開始を要求したものと仮定する。

【００４７】表４

仮説：ApplMgrは既にＪＶＭで実行中である。ApplMgrにより、アプリケーションクラスが予めロードされている。

結果：所望のApplicationが実行を開始する。

エージェント：

User＝ApplCilentに対し、ノード「ＨＰ」でアプリケーション「test」を開始するよう命令。

ApplClient＝Applicationsを開始するホスト側プログラム

ClientProtocol＝クライアントのためのネットワーク通信を処理するホスト側クラス

ServerProtocol＝ApplMgrネットワーク通信を処理するサーバ側クラス

ApplMgr＝アプリケーションマネージャクラス

Application＝開始すべきアプリケーションをモデル化するアプリケーションクラス

ApplBase＝開始中のアプリケーション・インスタンス

図7に示すように、開始動作は、クライアントユーザクラスApplClientU５１６等のクライアントユーザアプリケーションからクライアントアプリケーションApplClient５０８へのユーザ要求７０２によって開始する。この例では、クライアントユーザクラスApplClientU５１６は、「Ｂ」キャラクタ（「begin（開始）」に対応）と、それに続くアプリケーション名「test」と、更にそれに続くクラスデータが送信すべきネットワークノード「ＨＰ」とを送信するプロトコルに従うことにより、開始要求を生成する。要求７０２により、ApplClient５０８が実行され、ノード「ＨＰ」でアプリケーション「test」が開始される。ApplClient５０８は、ClientProtocol.startAppl()メソッドを呼出してアプリケーションを開始する（７０４）。ClientProtocol５０６は、ノード「ＨＰ」におけるServerProtocol５０４へのネットワークコネクションを形成し、開始メッセージを送信する（７０６）。ServerProtocol５０４は、開始要求をデコードし、その後ApplMgr.startAppl()メソッドを呼出す（７０８）。ApplMgr５０２は、適切なApplication５１０を配置し、その後Application５１０にstartAppl()メソッドを呼出す（７１０）。Application５１０は、新たなThread（スレッド）５１８を生成（７１２）してそれ自身のrun()（実行）メソッドを実行する（７１４）。run()メソッド内では、ApplBase５１４のメインメソッドが呼出され（７１６）、種々のパラメータが渡される。Application５１０にいかなるパラメータも渡されない場合、run()メソッドを直接に呼出すことができる。そして、Application５１０は、アプリケーションインスタンスの状態を「実行中」に変更する（７１８）。

【００４８】他の実施の形態では、ApplClassオブジェクト（すなわち、ClassLoader５１２）が、Application５１０のインスタンスをモデル化するApplicationクラスを管理する。

【００４９】図8は、アプリケーションの停止を実行するためのクラス間の通信を示すシステム５００のブロック図である。表５は、初期の仮説、停止動作の結果、および停止を実行するために必要なエージェントを指定している。ここに示すように、アプリケーションマネージャクラスApplMgrは組込み機器２０のＪＶＭ２２において実行中であり、かつ、ユーザが、そのユーザのマシンで実行中のApplClientに対し組込み機器２０でアプリケーションを停止するよう命令することにより停止を要求したものと仮定する。

【００５０】表５

仮説：ApplMgrは、既にＪＶＭにおいて実行中である。ApplMgrによりアプリケーションクラスが予めロードされている。アプリケーションは既に実行中である。

結果：所望のApplicationが実行を停止する。
エージェント：
User＝ApplClientに対しノードＨＰでのアプリケーションtestの停止を命令
ApplClient＝Applicationsを開始するホスト側プログラム
ClientProtocol＝クライアントに対するネットワーク通信を処理するホスト側クラス
ServerProtocol＝ApplMgrネットワーク通信を処理するサーバ側クラス
ApplMgr＝アプリケーションマネージャクラス
Application＝停止すべきApplBaseインスタンスを含むアプリケーションクラス
ApplBase＝停止すべき実行中のアプリケーションインスタンス
図８に示すように、停止動作は、クライアントユーザクラスApplClientU５１６等のクライアントユーザプリケーションからクライアントアプリケーションApplClient５０８へのユーザ要求８０２で開始する。この例では、クライアントユーザクラスApplClientU５１６は、「Ｅ」キャラクタ（「end（終了）」に対応）と、それに続くアプリケーション名「test」と、更にそれに続くクラスデータを送信すべきネットワークノード「ＨＰ」とを送信するプロトコルに従って、停止要求を生成する。ユーザ要求８０２により、ApplClient５０８が実行され、ノード「ＨＰ」でのアプリケーション「test」の実行が停止する。ApplClient５０８は、ClientProtocol.stopAppl()メソッドを呼出してアプリケーションを停止する（８０４）。ClientProtocol５０６は、ノード「ＨＰ」でのServerProtocol５０４に対するネットワークコネクションを形成し、停止メッセージを送信する（８０６）。ServerProtocol５０４は、停止要求をデコードし、その後ApplMgr.stopAppl()メソッドを呼出す（８０８）。ApplMgr５０２は、適切なApplication５１０を配置し、その後Application５１０にstopAppl()メソッドを呼出す（８１０）。Application５１０は、stopAppl()に渡された名前によってApplBase５１４インスタンスを配置し、そのterminate()（終了）メソッドを呼出す（８１２）。そして、アプリケーションは、Thread.join()を用いてApplBaseが終了したことを確実にする。Application５１０は、ApplBase５１４インスタンスの状態を「終了」に変更する（８１４）。
【００５１】アプリケーションはそれ自身のスレッドによって実行中であるため、ApplBase５１４インスタンスの停止は、非同期イベントである。アプリケーションの停止を、実際には待たないことが望ましい。それは、アプリケーションが、その終了フラグをポーリングしない場合、停止しないためである。従って、実現例では、その代りに、タイムアウトを用いてjoin()（ジョイン）メソッドを使用し、stop()を呼出すことにより強制的に終了してもよい。
【００５２】図９は、Application５１０のインスタンスの実行中にローメモリまたはノーメモリ状態が発生した場合のクラス間の通信を示すシステム５００のブロック図である。表６は、初期の仮説、アウトオブメモリ状態の結果、およびその状態の処理を実行するために必要なエージェントを指定している。ここに示すように、アプリケーションマネージャクラスApplMgr５０２は組込み機器２０のＪＶＭ２２において実行中であり、１つまたは複数のアプリケーションクラスがロードされ実行中であると仮定する。
【００５３】表６
仮説：ApplMgrは、既にＪＶＭにおいて実行中である。１つまたは複数のアプリケーションクラスがロードされ実行中である。
結果：ＪＶＭおよびApplMgrは、実行を継続し、ユーザアプリケーションクラスおよびインスタンスキャッシュを管理することによりメモリを解放する。
エージェント：
User＝ApplClientに対し、ノード「ＨＰ」でのアプリケーション「stop」を停止するよう命令
ApplClient＝Applicationsを開始するホスト側プログラム
ClientProtocol＝クライアントに対するネットワーク通信を処理するホスト側クラス
ServerProtocol＝ApplMgrネットワーク通信を処理するサーバ側クラス
ApplMgr＝アプリケーションマネージャクラス
Application＝停止すべきApplBaseインスタンスを含むアプリケーションクラス
ApplBase＝停止すべき実行中のアプリケーションインスタンス
【００５４】図９に示すように、実行中のApplication５１０のインスタンスApplBase５１４は、OutOfMemoryErrorを生成する（９０２）。好ましくは、OutOfMemoryErrorは、Application.run()メソッド内のメモリ管理ハンドラ２７によって処理される（９０４）。メモリ管理ハンドラ２７は、ApplMgr.ourOfMemory()メソッドを呼出す（９０６）ことにより、ApplMgr５０２に対しその状態を通知する。そして、Application５１０は、run()から戻り、ApplBase５１４スレッドを停止する（９０８）。ApplMgr５０２は、そのアプリケーションキャッシュ５２を通して、Application.applCount()メソッドを呼出す（９０８）ことにより実行中のApplBasesを有していないクラスを探す。ApplBasesを実行していないそれらアプリケーションは、freeMemory()と命令されれ（９１０）、それにより、アプリケーションキャッシュ５２からそれらのクラスをダンプし、すべてのオブジェクト参照をナルに設定する。その後、ApplMgr５０２は、そのアプリケーションキャッシュ５２からそれらア

プリケーションを除去し（９１２）、Runtime.gc()を呼出してガーベッジ・コレクションによって参照されないオブジェクトが収集されるようにする（９１４）。これにより、メモリが解放される。

【００５５】以上、本発明の実施例について詳述したが、以下、本発明の各実施態様の例を示す。

【００５６】（実施態様１）組込み機器（２０）において１つ以上のアプリケーション（２６ａ、２６ｂ、２６ｃ）を管理するアプリケーションマネージャ（２４）であって、前記組込み機器（２０）が、クラスオブジェクト（２８ａ、２８ｂ）を格納するアプリケーションキャッシュ（５２）と、ネットワーク（１４）を介してクライアント（１２）と通信するネットワークインタフェース（２５）と、該組込み機器（２０）にインストールされ該組込み機器（２０）上で実行されるＪａｖａ仮想マシン（ＪＶＭ）（２２）と、を備えるアプリケーションマネージャ（２４）において、（ａ）エントリを有し、前記エントリの各々が、前記組込み機器（２０）に現在ロードされているクラスオブジェクト（２８ａ、２８ｂ）を識別するクラスオブジェクトリスト（２３）と、（ｂ）前記ネットワークインタフェース（２５）を介して前記クライアント（１２）から受信するアプリケーションクラス（２８ａ、２８ｂ）をロードし、前記アプリケーションクラスに対する新たなクラスオブジェクト（２８ａ、２８ｂ）を生成し、前記アプリケーションキャッシュ（５２）に前記新たなクラスオブジェクト（２８ａ、２８ｂ）を格納し、前記新たなクラスオブジェクト（２８ａ、２８ｂ）を前記アプリケーションキャッシュ（５２）にロードされているものとして識別するために、前記クラスオブジェクトリスト（２３）に新たなエントリを付加するクラスローダメソッド（loadAppl()、loadAndInit()）と、を具備するアプリケーションマネージャ（２４）。

【００５７】（実施態様２）各クラスオブジェクト（２８ａ、２８ｂ）は、前記クラスオブジェクトリスト（２３）で識別されるものであり、エントリを有し、前記エントリの各々が、前記組込み機器（２０）上で現在インスタンス生成されている前記クラスオブジェクトのインスタンス（３０ａ、３０ｂ、３０ｃ）を識別するインスタンスリスト（２９ａ、２９ｂ）を具備する、実施態様１記載のアプリケーションマネージャ（２４）。

【００５８】（実施態様３）前記インスタンスリスト（２９ａ、２９ｂ）の各エントリは、該エントリに対応する前記インスタンス（３０ａ、３０ｂ、３０ｃ）の現実行状態を示す実行状態属性を有する、実施態様２記載のアプリケーションマネージャ（２４）。

【００５９】（実施態様４）開始すべきインスタンス（３０ａ、３０ｂ、３０ｃ）が存在しない場合、クラスオブジェクト（２８ａ、２８ｂ）の新たなインスタンス（３０ａ、３０ｂ、３０ｃ）を生成して、前記新たなインスタンス（３０ａ、３０ｂ、３０ｃ）を前記クラスオブジェクト（２８ａ、２８ｂ）の前記インスタンスリスト（２９ａ、２９ｂ）に付加し、前記開始すべきインスタンスが実行を開始するようにし、前記開始すべきインスタンス（３０ａ、３０ｂ、３０ｃ）が現在実行中であることを示すために、前記開始すべきインスタンスの前記インスタンスリスト（２９ａ、２９ｂ）の前記エントリに対応する前記実行状態属性を更新する、開始アプリケーションメソッド（Init()、Start()）を具備する、実施態様３記載のアプリケーションマネージャ（２４）。

【００６０】（実施態様５）前記組込み機器（２０）上で現在実行中であり、かつ、停止すべきであるインスタンス（３０ａ、３０ｂ、３０ｃ）が実行を停止するようにし、前記停止すべきインスタンス（３０ａ、３０ｂ、３０ｃ）が実行中でないことを示すために、前記停止すべきインスタンス（３０ａ、３０ｂ、３０ｃ）の前記インスタンスリスト（２９ａ、２９ｂ）の前記エントリに対応する前記実行状態属性を更新する停止アプリケーションメソッド（stopAppl()）を具備する、実施態様４記載のアプリケーションマネージャ（２４）。

【００６１】（実施態様６）ローメモリまたはアウトオブメモリ状態（OutOfMemoryError()）の検出に対応して、前記アプリケーションキャッシュ（５２）にキャッシングされたクラスオブジェクト（２８ａ、２８ｂ）のうちアンロードすべきクラスオブジェクト（２８ａ、２８ｂ）を選択し、前記アプリケーションキャッシュ（５２）から前記選択したクラスオブジェクト（２８ａ、２８ｂ）をアンロードするメモリ管理ハンドラ（２７）を具備する、実施態様１から実施態様５記載のアプリケーションマネージャ（２４）。

【００６２】上に詳述したように、本発明は、アプリケーションマネージャおよびＡＰＩ仕様を提供し、電子機器または装置において、およびリソース制約が制限された環境を有する他の組込みシステムにおいて使用することが企図されたＪａｖａ仮想マシン内で実行される。本発明は、より優先度の高いアプリケーションのためにメモリを解放すべくメモリリソースが制約されることとなった場合に、将来の実行のためにアプリケーションをキャッシングおよび終了する柔軟性を提供する。

【００６３】以下に、付録Ａ及び付録Ｂを添付する。

付録Ａ

ＪａｖａアプリケーションマネージャＡＰＩ

```
import java.io.IOException;
public interface ApplMgr
```

```
{
// アプリケーション中心のメソッド
public string[ ] applClasses( ) throws IOException;
public string[ ] applications( ) throws IOException;
public string[ ] applInstances( ) throws IOException;
public void loadApplClass( String applName, String className,
            boolean mainClass, byte classData[ ] )
            throws IOException;
public String initAppl( String applName ) throws IOException;
// 動作するメインクラスでなければならない
public String loadAndInit( String applName, String className,
              byte classData[ ] ) throws IOException;
public String startAppl( String applName, String applId ) throws IOExcep
tion;
public void stopAppl( String applName, String applId ) throws IOExceptio
n;
public void unloadAppl( String applName ) throws IOException;
// メモリＡＰＩ
public long freeMemory( ) throws IOException;
public long totalMemory( ) throws IOException;
public double getFreeMemoryLimit( ) throws IOException;
public void setFreeMemoryLimit( double percent ) throws IOException;
public boolean getFreeAppsFirstPolicy( ) throws IOException;
public void setFreeAppsFirst( boolean val ) throws IOException;
}// interface ApplMgr 終了
```

【００６４】

付録Ｂ

```
ＲＭＩベースのアプリケーションマネージャインタフェースの簡単な例public i
nterface ApplMgr extends java.rmi.Remote
{
// アプリケーション管理  API
public void loadClass( String className, byte classData[ ], boolean main
 ) throws RemoteException ;
public Object startAppl( String className, String args[ ] ) throws Remot
eException ;
public void stopAppl( Object appl ) throws RemoteException;
unloadClass( String className );
//ロードされたクラスのリスト及び実行中のアプリケーション名の入手
public String[ ] enumerateApplClasses ( )  throws RemoteException ;
public String[ ] enumerateAppls( ) throws RemoteException;
// JVM情報の入手  - システム及びランタイムクラスを参照
public long freeMemory ( )  throws RemoteException;
public long totalMemory ( )  throws RemoteException;
public Properties getProperties ( )  throws RemoteException ;
}
```

【図面の簡単な説明】

【図１】本発明が動作するネットワークシステムのシステム図である。

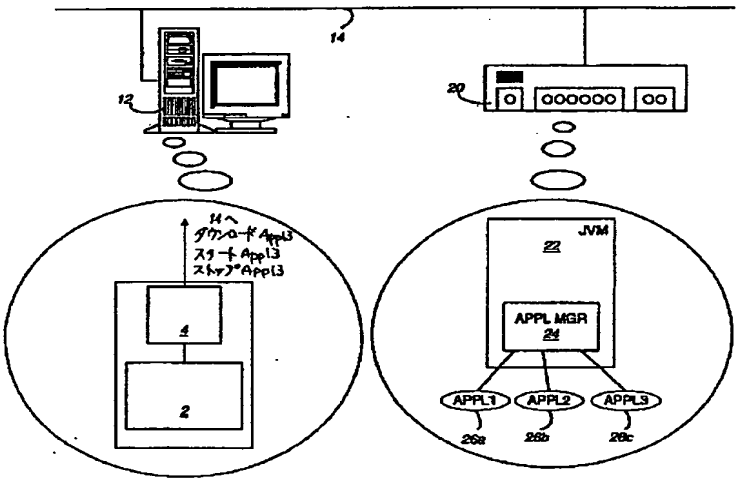
【図２】本発明が実行される組込み機器のブロック図である。

【図３】本発明に従って、アプリケーションが実行される過程を示す系統線図である。

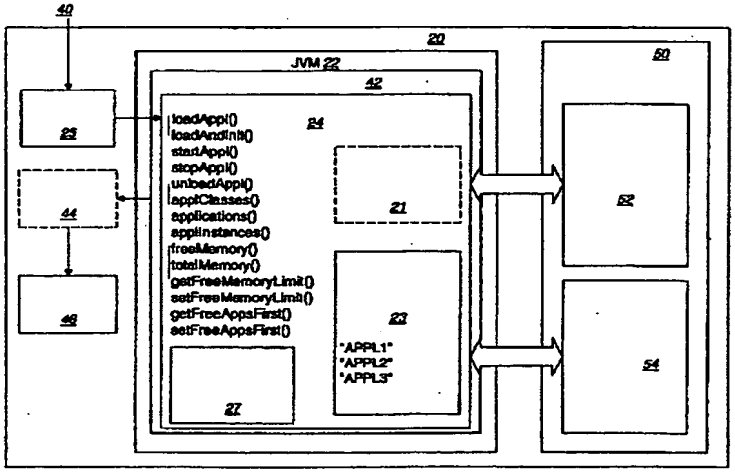
【図４】本発明に従って実現されるアプリケーションマネージャの１つの実施形態の動作フローチャートである。

【図５】本発明に従って実現されるアプリケーションマネージャの１つの実現例のクラスダイアグラムである。
【図６】ロード動作を実行するための図５に示すクラス間の通信を示す系統線図である。
【図７】開始動作を実行するための図５に示すクラス間の通信を示す系統線図である。
【図８】停止動作を実行するための図５に示すクラス間の通信を示す系統線図である。
【図９】アウトオブメモリ状態を処理するための図５に示すクラス間の通信を示す系統線図である。
【符号の説明】
１２：コンピュータシステム
１４：ネットワーク
２０：組込み機器
２３：アプリケーションリスト
２４：アプリケーションマネージャ
２５：ネットワークインタフェース
２６ａ、２６ｂ、２６ｃ：アプリケーション
２７：メモリ管理ハンドラ
２８ａ、２８ｂ：クラスオブジェクト
２９ａ、２９ｂ：インスタンスリスト
３０ａ、３０ｂ、３０ｃ：アプリケーションオブジェクト
５２：アプリケーションキャッシュ

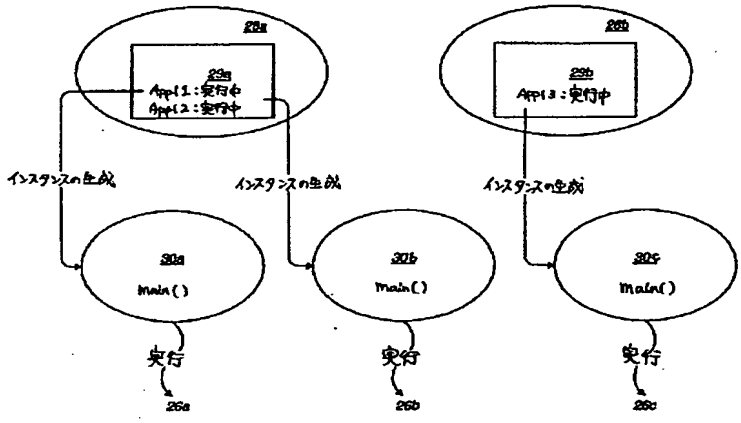
【図１】



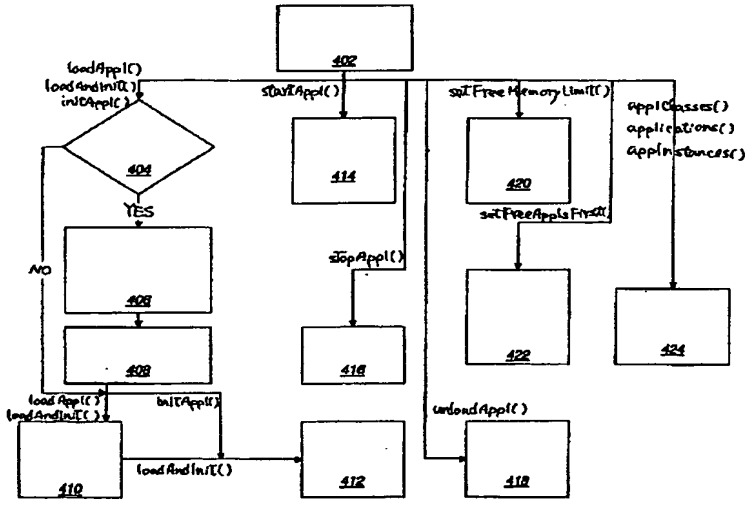
【図２】



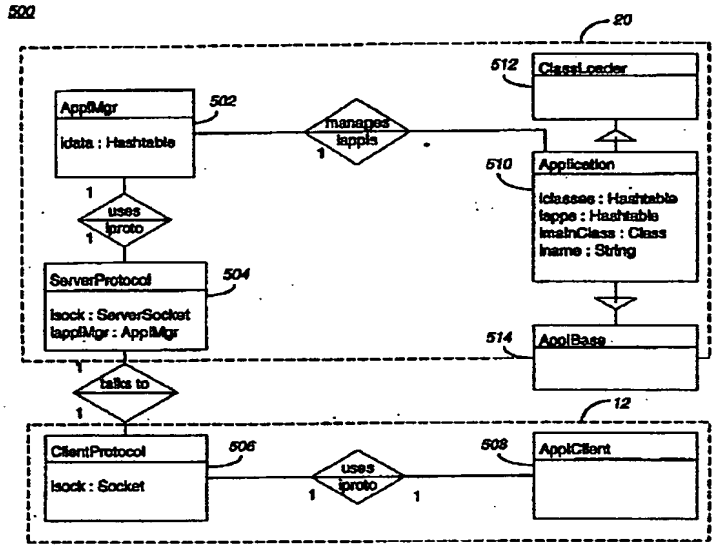
【図3】


28a
29a
Appl1:実行中
Appl2:実行中
28b
29b
Appl3:実行中
インスタンスの生成
インスタンスの生成
インスタンスの生成
30a
main()
30b
main()
30c
main()
実行
実行
実行
26a
26b
26c

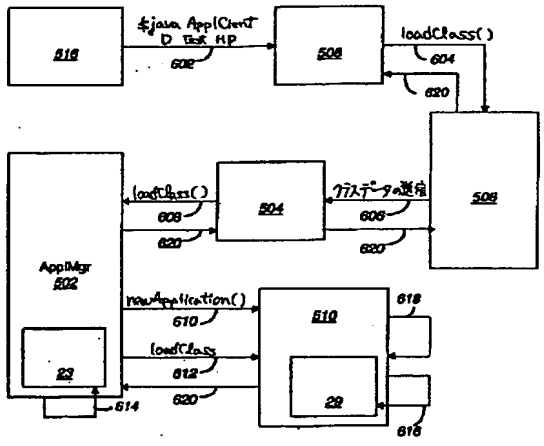
【図4】


402
loadAppl()
loadAndInit()
initAppl()
404
YES
NO
406
408
loadAppl()
loadAndInit()
410
initAppl()
loadAndInit()
412
startAppl()
414
stopAppl()
416
setFreeMemoryLimit()
420
setFreeApplsFirst()
422
unloadAppl()
418
applClasses()
applications()
applnstances()
424

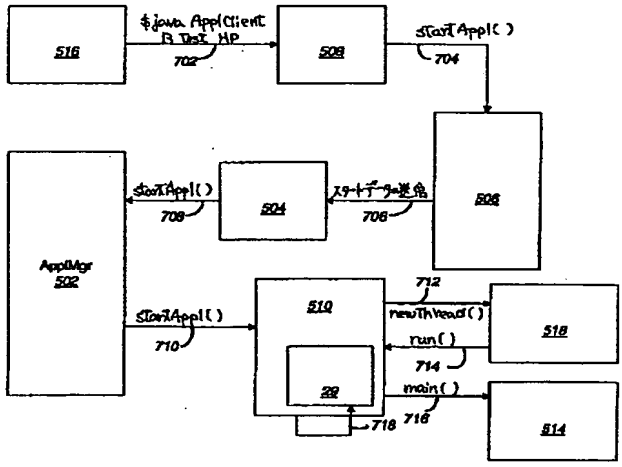
【図5】


502
20
512
ClassLoader
AppMgr
idata : Hashtable
502
manages
lappls
1
1
510
Application
iclasses : Hashtable
lappe : Hashtable
imainClass : Class
iname : String
1
uses
iproto
1
ServerProtocol
504
isock : ServerSocket
lappMgr : AppMgr
514
AppBase
talks to
1
12
ClientProtocol
506
isock : Socket
uses
iproto
1
1
508
AppClient

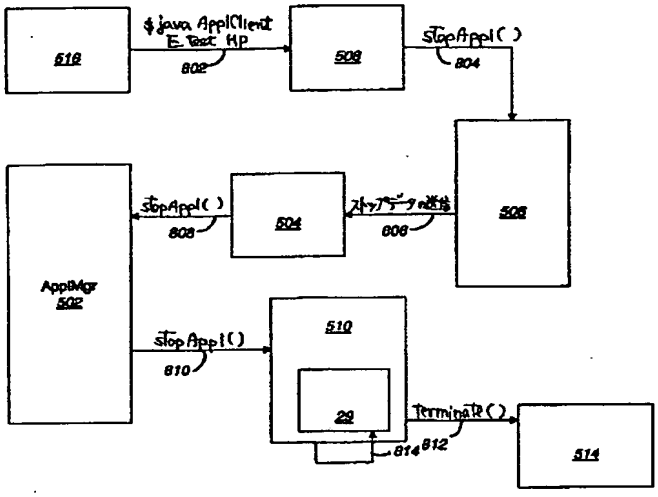
【図6】


516
$java AppClient
D Test HP
602
508
loadClass()
604
620
AppMgr
502
23
614
loadClass()
608
620
504
クラスデータ送信
606
620
508
newApplication()
610
loadClass
612
620
510
29
618
618

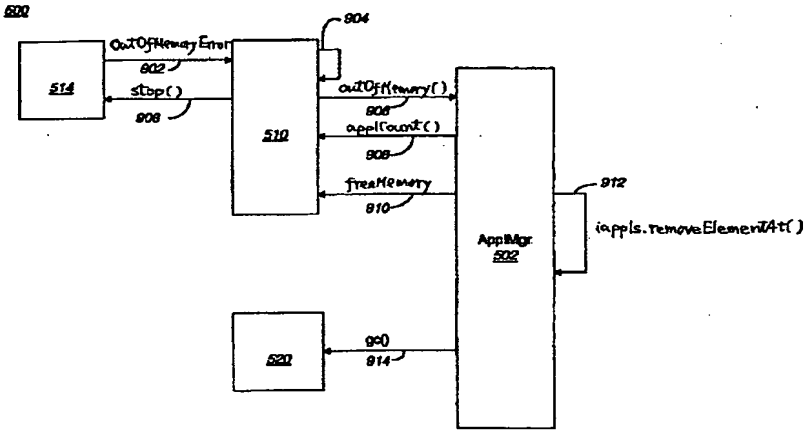
【図7】


516
$java AppClient
D Test HP
702
508
startAppl()
704
AppMgr
502
startAppl()
708
504
スタートデータ送信
706
508
startAppl()
710
510
29
712
newThread()
run()
714
main()
716
718
518
514

【図8】


510
$ Java AppClient
E. Test HP
802
508
stopAppl( )
804
AppMgr
502
stopAppl( )
808
504
806
510
stopAppl( )
810
29
terminate( )
814
812
514

【図9】


900
514
outOfMemoryError
902
stop( )
908
904
510
outOfMemory( )
906
appCount( )
908
freeMemory
910
AppMgr
502
912
iappls.removeElementAt( )
520
gc()
914

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.